1 Sc LEVEL II

ADA084293

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

DDC FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DTIC ELECTE MAY 19 1980 S D E

80 5 19 116

TR-794
DAAG-53-76C-0138

July 1979

A PATH-LENGTH DISTANCE TRANSFORM FOR QUADTREES

Michael Shneier
Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

A distance measure is defined for a quadtree representation of a binary image. An algorithm is presented that calculates the distance from the center of each BLACK node to the border of the nearest WHITE node. The distance is defined as the path length from the center of the BLACK node, through the center of intervening BLACK nodes, to the WHITE border. The worst case average execution time is shown to be proportional to the product of the log of the image diameter and the number of blocks in the image.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The author has benefitted greatly from discussions with Charles R. Dyer, Azriel Rosenfeld, and Hanan Samet.

1. Introduction

The size of a region in an image and distances between subregions of an image are useful geometric properties for describing region shape. In particular, the distance of interior points from the border is a useful measure for operations such as thinning and finding skeletons of regions, [6].

Most algorithms for finding the distance from a point inside a region to the border have involved multiple passes over the data, each pass incrementing a local counter denoting the distance of the associated point in the image, ~~[6]~~. ~~We~~ present ~~here~~ an algorithm that computes the distance function in a single pass over an image represented by a quadtree. The algorithm has the advantage over previous algorithms that blocks of a region are processed at once, instead of individual points, and distance information is very quickly propagated to the interior of the region. In fact, we show that every block in the region is separated from the boundary by a sequence of blocks whose total width is not greater than the size of the block.

Recent research on quadtrees [2-5,7-14] has produced interesting results in several areas of image processing. This paper continues the investigation of the usefulness of quadtrees as a representation for binary images.

2. Definitions and notation

We assume that the given image is a 2^n by 2^n binary array of unit square "pixels". A quadtree is constructed from such an image by successively dividing it into quadrants, subquadrants, etc. until blocks (possibly single pixels) are obtained that are made up entirely of either 0's or 1's. The process is represented by a tree whose root node represents the entire array, and in which each non-terminal node has four sons, corresponding to the quadrants of its father. The terminal nodes are those corresponding to blocks of the array that need not be subdivided further. Figure 1a shows a sample image divided into quadrants and subquadrants. Figure 1b shows the corresponding quadtree.

Let each node in a quadtree be stored as a record containing eleven fields. The first five fields contain pointers to the node's father, and to its four sons, labeled NW, NE, SW, and SE. Given a node P and a son I, these fields are referred to as FATHER(P) and SON(P,I), respectively. The son links are defined only for non-terminal nodes. The sixth field, named NODETYPE, describes the contents of the block of the array represented by the node--i.e. WHITE if the block contains no 1's, BLACK if the block contains only 1's, and GRAY if it contains both 0's and 1's. BLACK and WHITE nodes are terminal nodes, while GRAY nodes are non-terminal nodes.

The other five fields are used for storing the distance information. Each node will have a distance (DIST) representing the distance from the center of the block represented by the node to the border of the nearest WHITE block. The last four fields are for storing information concerning the distance through the node along paths entering in each of the directions N, S, W, and E. Each GRAY node stores the distance from each of its sides to the nearest WHITE block. The distance for a node P and a side T is referenced by SIDEDIST(P,T). For WHITE nodes, the SIDEDISTS are all zero, while for BLACK nodes they will all store the minimum distance from the node to the nearest WHITE node. All distances have initial value UNDEF. See Section 3 for a more precise definition of distance.

A number of functions will be useful in manipulating the representation. It will sometimes be necessary to know spatial relationships between quadrants and sides of quadrants. The function OPSIDE(B) corresponds to the side facing side B. For example, OPSIDE(N) = S, OPSIDE(W) = E. The predicate ADJ(B,I) is true if and only if quadrant I is adjacent to side B of the node's block; e.g. ADJ(N,NW) is true. REFLECT(B,I) yields the quadrant adjacent to quadrant I along side B of the block represented by I; e.g. REFLECT(N,NW) = SW, REFLECT(E,NW) = NE.

Given a quadtree corresponding to a 2^n by 2^n array, we say that the root is at level n, and that a node at level i

is at a distance n-i from the root; i.e. from a node at level i we must ascend n-i FATHER links to reach the root of the tree. The furthest node from the root is at level ≥ 0. A node at level 0 corresponds to a single pixel in the image, while a node at level i corresponds to a 2^i by 2^i block of the image.

3. Distance

There are several metrics that are commonly used in image processing. These metrics, or distance functions, take pairs of points into non-negative numbers representing a distance between the points. Examples of these metrics are the Euclidean distance, d_e, the city block distance, d_4, and the maximum value, or chessboard distance, d_8. For two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$,

$$d_e(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

$$d_4(p,q) = |x_p - x_q| + |y_p - y_q|$$

$$d_8(p,q) = \max(|x_p - x_q|, |y_p - y_q|)$$

Of these metrics, Samet [13] gives arguments in favor of using the chessboard metric for quadtree applications. It should be noted, however, that there is another class of distance measures that are applicable to quadtrees. These measures depend on the structure of the tree and on paths through the tree, rather than on the underlying image represented by the tree. In some applications, these alternative distance measures may be preferable to the image-based definitions.

Samet [13] gives a bottom-up algorithm for finding the chessboard distance from the center of each BLACK node in a quadtree to the nearest point on the border of a WHITE node.

This paper presents a top-down algorithm, using a simplified form of the distance measure. A straightforward extension to the algorithm can be made to give a top-down version of the chessboard distance algorithm.

We define a distance transform T for a quadtree to be a function that yields, for each BLACK node in the quadtree, the distance from the center of the block represented by the node to the border of the closest WHITE block. The transformation used here is based on the chessboard metric, but depends also on finding paths through the quadtree.

More formally, let x be the center of a BLACK block B, y be the center of a WHITE block W, and let P_{xz} denote the set of paths from x to a point z on the border of the block centered at y.

A <u>path</u> is defined as a sequence of blocks which are each adjacent along a side to their predecessor and successor blocks. There are no diagonal steps in a path, and the path runs through the center of each block. The <u>length</u> of a path P_{xz} through a sequence of blocks $b_1, b_2, \ldots, b_n$, where x is the center of b_1, and z is on the border of b_n, is defined as follows: it starts at x, the center of block b_1, passes through all the intermediate blocks, and ends at the border of block b_n. If the centers of blocks $b_1, \ldots, b_n$ are points $x_1, \ldots, x_n$, the path length is

$$L(P_{pq}) = \sum_{i=1}^{n-2} d_8(x_{i+1}, x_i) + 1/2(\text{size of } b_{n-1})$$

$$= 1/2(\text{size of } b_1) + \sum_{i=2}^{n-1} (\text{size of } b_i)$$

The distance transform, T(B), is defined as the minimum path length from a point x at the center of the BLACK block B to some point z on the border of the closest WHITE block. That is,

$$F(B,W) = \min_{z} L(P_{xz})$$

$$T(B) = \min_{W} F(B,W)$$

We define T of a WHITE block or a GRAY block to be zero. The distance is not defined in terms of a center to center distance in order to avoid a bias against large WHITE neighbors.

Samet [13] defined a different transform based on the same underlying metric. Istead of requiring a minimum path, however, he defined the distance as the value of the chessboard metric applied between the center of a BLACK block and the nearest point on a WHITE border. The transform defined here is guaranteed always to give distance values greater than or equal to those of Samet. This is a direct result of the triangle inequality property of the underlying metric. For example, the distance of block 21 of Figure 1a would be calculated along the path from the center of block 21 to the center of block 16 (or block 18 or 20) and then to the WHITE border, a distance of $1\frac{1}{2}$ units. Using the transform proposed by Samet, the

distance would be the chessboard distance from the center of block 21 to the corner of block 15 or 17, a distance of 1/2 unit. The difference in this example is due to the fact that the transform given here does not allow diagonal steps. This restriction gives rise to defects in the distance measure. These are discussed further in Section 6.

Notice that the chessboard distance metric can result in gaps being left in the shortest path. For example, in Figure 2, the distance from the BLACK node A to the WHITE node B is simply the horizontal distance from the center of A to the border of the GRAY node to which B belongs. The vertical distance across node C is ignored because it is less than the horizontal distance (see the definition of the chessboard metric).

4. The algorithm

The algorithm involves a top-down traversal of the quadtree. For each node it calculates the distance transform and also stores information that is helpful in processing other nodes. This extra information concerns the shortest path through the node to a WHITE border, that enters the node from each of the four sides. The calculations performed when visiting a node depend on the type of the node. For WHITE nodes, the distance is immediately set to zero. For BLACK nodes, the distance is a function of the path lengths through its neighbors of the same size, and for GRAY nodes the calculations performed involve looking at the sons of the node.

The distance of a BLACK node depends on the shortest paths through neighbors of its own size (a neighbor may be GRAY if the leaf nodes are smaller than the BLACK node). For a 2^k by 2^k BLACK block, P, the distance is the radius of $P(2^{k-1})$ plus the shortest distance from the border of P to the border of a WHITE node. That is,

$$
\begin{aligned}
\mathrm{DIST}(P) = 2^{k-1} + \min\{&\text{distance from S border of N neighbor},\\
&\text{distance from W border of E neighbor},\\
&\text{distance from N border of S neighbor},\\
&\text{distance from E border of W neighbor}\}.
\end{aligned}
$$

For example, node 21 of Figure 1a has a distance of

2^{0-1} (the radius of the 1 by 1 node)

+ min{distance from S border of 16, distance from W border of 20, distance from N border of 18, distance from E border of 22}

$= 1/2 + \min\{1,1,1,3\}$

$= 1\frac{1}{2}$

We will show shortly, however, that it is not necessary to look at node 22 in performing the calculation because node 22 is too large to be included in the shortest path.

In addition to the distance, each node establishes four SIDEDISTs. These are the distances to the nearest WHITE node assuming that the path enters the node at a given side. For example, when node 22 of Figure 1a attempts to find its distance, it looks at its neighbors, nodes f, g, i, j (GRAY nodes are labelled with letters, BLACK and WHITE nodes with numbers; see Figure 1b). For node f, the path to a WHITE node from node 22 must enter from the south, so the interesting SIDEDIST is the south SIDEDIST. Similarly, the distance along a path from node 22 going east would be calculated using the west SIDEDIST of node i. In fact, the length of the path would be the radius of node 22 plus the value of the west SIDEDST of node i--i.e. $2^{1-1} + 1 = 2$. Note that the shortest distance to a WHITE node starting at the western border of node i is the width of node 18, a distance of 1 unit.

For BLACK nodes, the SIDEDISTS are all equal to the shortest distance from the center of the node to a WHITE border, plus

the radius of the node. This corresponds to the requirement that any path through the node must pass through the center of the block, and must follow the shortest path to a WHITE border. For WHITE nodes, the SIDEDISTS are all zero.

The bulk of the processing of GRAY nodes is concerned with establishing their SIDEDIST values. The DIST value of a GRAY node is defined to be zero. The SIDEDISTs of GRAY nodes may differ, but they are always less than the size of the node, because there is guaranteed to be a WHITE block contained within the GRAY block.

The SIDEDISTs for a GRAY node are established by looking at the sons of the node. If the sons are terminal nodes then their distances and positions in the block described by the GRAY node are used to calculate the SIDEDISTs. If a son is also a GRAY node, its SIDEDISTs in the appropriate directions are used instead. For example, node f (blocks 2, 3, 4, and 5) of Figure 1a has the following SIDEDISTs: SIDEDIST(N) = 0, SIDEDIST(W) = 0, SIDEDIST(S) = 1, and SIDEDIST(E) = 0.

The algorithm starts at the root of the tree, whose level is given. The level is the log of the diameter of the image--i.e. n for a 2^n by 2^n image. If the root node is a terminal node the distance is calculated as 0 for a WHITE node and as UNDEF for a BLACK node, and the algorithm terminates. If the root node is a GRAY node, the distances of its sons are

established by recursively applying the algorithm to the sons.

Notice that there is a possibility that a cycle will arise in the algorithm as stated. A BLACK node must first establish the distances of its neighbors before it can calculate its own distance. Suppose, however, that all of its neighbors are BLACK (e.g. node 21 in Figure 1a). When these BLACK nodes attempt to find <u>their</u> distances, they must interrogate the original calling node, thus setting up a cycle. We will show that the shortest path from the called node to a WHITE border cannot pass through the calling node. This justifies prohibiting a BLACK called node from looking at its calling node in calculating its distance, and prevents cycles from developing. We first prove a result that justifies the top-down nature of the algorithm.

<u>Lemma</u>

The minimum distance path from a node to the border of the nearest WHITE node does not pass through nodes larger than the starting node.

<u>Proof</u>

If the starting node is WHITE or GRAY, the result is trivial. Suppose the starting node is a BLACK node of size 2^k by 2^k, and the path to the nearest WHITE node passes through a node of size 2^{k+1} by 2^{k+1}. Then the minimum distance required in traversing this 2^{k+1} by 2^{k+1} node is 2^{k+1}, excluding the

radius of the starting node. We show below, however, that the maximum length from the starting 2^k by 2^k node to the nearest WHITE border is $2^{k+1}-1$. From this the result follows immediately.

Two of the neighbors of each node are in the same block (i.e. have the same father) as that node. If we consider the fact that the four sons of the same father cannot be leaves of the same type (we can exclude GRAY nodes because the starting node is known to be BLACK) it is apparent that within the 2^{k+1} by 2^{k+1} block made up by the four sons, there must be at least one WHITE block, or else the father would have been a leaf node. Consider the case when this WHITE node is as far as possible from the starting node (Figure 3).

The shortest path to this WHITE node, excluding as before the radius of the starting node, is

$$
\begin{array}{ll}
\ 2^k & \text{(for the } 2^k \text{ by } 2^k \text{ intervening block)} \\
+\ 2^{k-1} & \text{(for the } 2^{k-1} \text{ by } 2^{k-1} \text{ intervening block)} \\
+\ \ldots & \text{(all the } 2^{k-i} \text{ by } 2^{k-i} \text{ intervening blocks, } i=1,2,\ldots,k-2) \\
+\ 2^0 & \text{(for the block adjacent to the WHITE block)} \\
=\ \sum_{i=0}^{k} 2^i = 2^{k+1}-1 &
\end{array}
$$

$\square$

We are now ready to prove that cycles in the algorithm can be prevented without jeopardizing the distance calculations.

Theorem

When a node is interrogated about its distance to the border of the nearest WHITE node, that distance does not depend on the interrogating node.

Proof

The result is immediate for WHITE and GRAY nodes. The only time a BLACK node is interrogated is when the calling node is surrounded by BLACK nodes of its own size, say 2^k by 2^k, or larger. In this case, the minimum distance from the called node, also of size 2^k by 2^k, to a WHITE border in the direction of the calling node is at least 2^{k+1} (two BLACK nodes lie on that path before the first non-black node). For example, consider node 21 in Figure 1a. This node is surrounded by BLACK nodes. When it calls, say, node 20, the minimal distance path from node 20 through node 21 must pass through both node 21 and node 16 (or node 18). The lemma, however, showed that the maximum distance from any node of size 2^k by 2^k is less than 2^{k+1}, so this path cannot possibly be the shortest.

$\square$

Example

Figure 1a shows a block decomposition of a region for which Figure 1b is the corresponding quadtree representation. Figure 1c shows the distances calculated for the leaf nodes of the quadtree. The blocks of Figure 1a are numbered in the order in which their distances are calculated. The quadtree of Figure

1b reflects the labeling in Figure 1a; terminal nodes are labeled with numbers, and non-terminal nodes with letters.

The algorithm starts with a pointer to the root of the tree (node a), a level number (in this case 3) indicating the size of the picture (2^3 by 2^3), and a direction for the node (UNDEF, because there is no calling node). The starting node is GRAY, and it immediately tries to establish the distances of its sons, nodes b, c, d, and e.

Node b is also GRAY, and must establish the distances of its sons, nodes 1, f, g, and 22. Node 1 is WHITE, and is immediately assigned a distance of 0. Node f is GRAY, and calls its sons, nodes 2, 3, 4, and 5. These nodes are all at the lowest level in the tree. Nodes 2 and 3 are WHITE and are assigned zero distance. Nodes 4 and 5 are both adjacent to WHITE nodes and are assigned a distance equal to their radius--i.e. the distance from their centers to their borders, in this case 1/2. Their SIDEDISTs are equal to the size of the node, i.e. 1, because a path through them to the WHITE border would have to traverse the whole node.

The calling node, f, is now able to establish its SIDEDISTs. They are 0 for N, W, and E, and 1 for S.

A similar process establishes the distances of node g and its sons. The fourth son of b, node 22, is BLACK. It must establish the distances of its neighbors in order to find its own distance. The neighbors of node 22 are nodes f(N), g(W),

j(S) and i(E). All these nodes are GRAY. Two of them, f and g, already have their distances established, so only j and i need be processed. The treatment of node j and its sons is straightforward. The call of i, however, leads to a call of the sons of i, and of particular interest, to the son 21 of i. This node has four BLACK neighbors, and their distances must be established in order to establish that of 21. Because of the lemma, it is known that node 22 need not be examined (because node 22 is a larger BLACK node). This leaves nodes 16, 18, and 20, all of which are adjacent to WHITE nodes and have distances of 1/2. The distance of node 21 is the minimum of the distances of its neighbors, plus the distance from the center of the minimal neighbor to its own center. This is the same as the SIDEDIST of the minimal neighbor on the side abutting node 21, plus the radius of node 21, in this case, $1/2 + 1 = 1\frac{1}{2}$. Node 21 has its SIDEDISTs set equal to its distance plus its radius--i.e., the distance to pass right through the node to the nearest WHITE border. In this case, the SIDEDISTs are set to 2.

The rest of the distances are established in a similar way; the complete sequence of events is portrayed in Table 1.

The algorithm is given below. It is called with a pointer to the root of the quadtree (TREE), a direction (DIRECTION) that has initial value UNDEF, and an integer LEVEL corresponding

to the image size of 2^{LEVEL} by 2^{LEVEL}. Note that whenever a GRAY node makes a recursive call of the function, the value of LEVEL is UNDEF because there is no preferred direction of the call. The direction is only of use in preventing the cycles that could arise when a BLACK node requests the distance of a BLACK neighbor.

```
integer procedure DISTANCE(TREE,DIRECTION,LEVEL);
/*  find the distance from the center of each BLACK node to
    the border of the nearest WHITE node.  The path followed
    goes through the centers of intervening BLACK nodes */.
begin
    node TREE,P,Q,R,NBRS(4);
    side DIRECTION,I,L:
    integer J,K;
    if null(TREE) then return(0);
    K = 10000; /* a large number */
    if DIST(TREE)≠ UNDEF then /* the node has already been visited */
        return (if DIRECTION ≠ UNDEF then
                    SIDEDIST(TREE,OPSIDE(DIRECTION))
                 else DIST(TREE));
    if NODETYPE(TREE) = WHITE then
        begin
            for I in {N,W,S,E} do
                SIDEDIST(TREE,I):=0;
            DIST(TREE):=0;
            return(0);
        end
    else if NODETYPE(TREE) = GRAY then
        begin
            for Q in {NW,NE,SW,SE} do
                DISTANCE(SON(TREE,Q),UNDEF,LEVEL-1);
            for P in {NW,NE,SW,SE} do
```

```
                    begin
                        for (Q,R,I) in {(NW,NE,N),(NE,SE,E),
                                        (SW,SE,S), (NW,SW,W)} do
                            SIDEDIST(TREE,P):=min(SIDEDIST(SON(TREE,Q),I),
                                                  SIDEDIST(SON(TREE,R),I))
                    end
               return (if DIRECTION ≠ UNDEF then
                        SIDEDIST(TREE,OPSIDE(DIRECTION))
                    else DIST(TREE));
          end
     else  /* BLACK node */
          begin
               for I in {N,S,W,E} do
                    begin
                         NBR[I]:=FIND_NEIGHBOR(TREE,I);
                         if NBR[I] ≠ NULL and NODETYPE(NBR[I])=WHITE then
                              begin
                                   DIST(TREE)=2↑(LEVEL-1);
                                   for L in {N,S,W,E} do
                                        SIDEDIST(TREE,L):=2↑LEVEL;
                                   return(SIDEDIST(TREE,'N'));
                              end;
                    end
               if all neighbors are NULL then
                    return(UNDEF) /* root is BLACK */
               else if all neighbors are BLACK or NULL nodes then
                    for I in {N,S,W,E} and I≠OPSIDE(DIRECTION)
                         and NBR[I]≠NULL do
```

```
                    begin
                        J:=DISTANCE(NBR[I],I,LEVEL);
                        if J<K then K:=J;
                    end
            else K:=min(adjacent side SIDEDISTs of
                        non-BLACK neighbors);
            DIST(TREE):=K+2↑(LEVEL-1);
            for I in {N,S,W,E} do
                  SIDEDIST(TREE,I):=K+2↑LEVEL;
            return(SIDEDIST(TREE,N));
          end
  end
```

1. Introduction

The size of a region in an image and distances between subregions of an image are useful geometric properties for describing region shape. In particular, the distance of interior points from the border is a useful measure for operations such as thinning and finding skeletons of regions [6].

Most algorithms for finding the distance from a point inside a region to the border have involved multiple passes over the data, each pass incrementing a local counter denoting the distance of the associated point in the image [6]. We present here an algorithm that computes the distance function in a single pass over an image represented by a quadtree. The algorithm has the advantage over previous algorithms that blocks of a region are processed at once, instead of individual points, and distance information is very quickly propagated to the interior of the region. In fact, we show that every block in the region is separated from the boundary by a sequence of blocks whose total width is not greater than the size of the block.

Recent research on quadtrees [2-5,7-14] has produced interesting results in several areas of image processing. This paper continues the investigation of the usefulness of quadtrees as a representation for binary images.

2. Definitions and notation

We assume that the given image is a 2^n by 2^n binary array of unit square "pixels". A quadtree is constructed from such an image by successively dividing it into quadrants, subquadrants, etc. until blocks (possibly single pixels) are obtained that are made up entirely of either 0's or 1's. The process is represented by a tree whose root node represents the entire array, and in which each non-terminal node has four sons, corresponding to the quadrants of its father. The terminal nodes are those corresponding to blocks of the array that need not be subdivided further. Figure 1a shows a sample image divided into quadrants and subquadrants. Figure 1b shows the corresponding quadtree.

Let each node in a quadtree be stored as a record containing eleven fields. The first five fields contain pointers to the node's father, and to its four sons, labeled NW, NE, SW, and SE. Given a node P and a son I, these fields are referred to as FATHER(P) and SON(P,I), respectively. The son links are defined only for non-terminal nodes. The sixth field, named NODETYPE, describes the contents of the block of the array represented by the node--i.e. WHITE if the block contains no 1's, BLACK if the block contains only 1's, and GRAY if it contains both 0's and 1's. BLACK and WHITE nodes are terminal nodes, while GRAY nodes are non-terminal nodes.

The other five fields are used for storing the distance information. Each node will have a distance (DIST) representing the distance from the center of the block represented by the node to the border of the nearest WHITE block. The last four fields are for storing information concerning the distance through the node along paths entering in each of the directions N, S, W, and E. Each GRAY node stores the distance from each of its sides to the nearest WHITE block. The distance for a node P and a side T is referenced by SIDEDIST(P,T). For WHITE nodes, the SIDEDISTS are all zero, while for BLACK nodes they will all store the minimum distance from the node to the nearest WHITE node. All distances have initial value UNDEF. See Section 3 for a more precise definition of distance.

A number of functions will be useful in manipulating the representation. It will sometimes be necessary to know spatial relationships between quadrants and sides of quadrants. The function OPSIDE(B) corresponds to the side facing side B. For example, OPSIDE(N) = S, OPSIDE(W) = E. The predicate ADJ(B,I) is true if and only if quadrant I is adjacent to side B of the node's block; e.g. ADJ(N,NW) is true. REFLECT(B,I) yields the quadrant adjacent to quadrant I along side B of the block represented by I; e.g. REFLECT(N,NW) = SW, REFLECT(E,NW) = NE.

Given a quadtree corresponding to a 2^n by 2^n array, we say that the root is at level n, and that a node at level i

is at a distance n-i from the root; i.e. from a node at level i we must ascend n-i FATHER links to reach the root of the tree. The furthest node from the root is at level ≥ 0. A node at level 0 corresponds to a single pixel in the image, while a node at level i corresponds to a 2^i by 2^i block of the image.

```
node procedure FIND_NEIGHBOR(P,S);
/* Given node P, return a node that is adjacent to side S of
   node P if the node is the same size as P, or a larger WHITE
   node.  Otherwise, return NULL */
begin
    node P,Q;
    side S;
    if not NULL(FATHER(P)) and ADJ(S,SONTYPE(P)) then
        /* find a common ancestor */
            Q:=FIND_NEIGHBOR(FATHER(P),S)
    else
            Q:=FATHER(P)
            /* follow reflected path to locate the neighbor */
    return (if not NULL(Q) and NODETYPE(Q)=GRAY then
            SON(Q,REFLECT(S,SONTYPE(P))
            else if NULL(Q) or NODETYPE(Q)=BLACK then NULL
                 else (Q)
end
```

5. Analysis

The running time of the path-length distance computation algorithm depends on the size of the quadtree and on the time spent finding neighbors of BLACK nodes. For WHITE and GRAY nodes there is no need to find neighbors, so it is only for the BLACK nodes that FIND_NEIGHBOR need be invoked. FIND_NEIGHBOR is called in succession on the N, S, W, and E neighbors of a BLACK node. If, at any stage, one of the neighbors is found to be WHITE, the remaining FIND_NEIGHBOR calls are not made.

If there are B BLACK nodes in the quadtree, at most 4B calls of FIND_NEIGHBOR will be made. Samet [7] has shown that the average time required to find a common ancestor of two neighboring nodes in a random quadtree is 2. That is the average path length to the common ancestor is 2. This implies that on the average, for each call of FIND_NEIGHBOR, 4 nodes are visited. So, if there are B BLACK nodes in the tree, the expected number of nodes visited in computing their distances is at worst $4 \cdot 4B = 16B$.

In addition, the top-down traversal of the tree visits each node once, and there are a maximum of $4Bn+1$ nodes in the quadtree (see Samet [8]). Thus, the total expected number of nodes visited in the course of the algorithm is at worst $4Bn+1+16B = 4B(n+r)+1$. That is, the average worst case execution time of the algorithm is proportional to the product of the number of BLACK nodes and the log of the diameter of the image.

6. Discussion

The algorithm described in this paper computes a minimum path length transform for a binary image represented by a quadtree. The algorithm involves a top-down tree traversal. This is in contrast to the bottom-up method described in [13], where a different notion of distance is used. The problem with that algorithm is that the work involved in traversing parts of the tree might have to be repeated for nodes that have the same closest WHITE neighbor. For instance, nodes A, B, and C in Figure 3 all have the same closest WHITE neighbor, and all share part of the same shortest path to that neighbor. This gives rise to a slightly higher estimate of the execution time of that algorithm over the one presented here. It should, however, be noted that the premises on which the analysis are based are not realistic. The assumption of a random quadtree, where any node is equally likely to be BLACK, WHITE, or GRAY, ignores the relationship between nodes of different levels in the tree. However, short of exhaustively enumerating all the quadtrees of fixed sizes, and calculating the likelihoods of the distributions of nodes, these assumptions may serve as rough estimates of performance.

An objection to the use of the algorithm presented here is its extra storage requirements. These requirements can be substantially reduced as follows. First, it is not necessary

to store any distance or SIDEDIST information in WHITE nodes because these values are zero by definition. For GRAY nodes, only the SIDEDIST values have any meaning, and the DIST value need not be stored. For BLACK nodes, the DIST value is essential, but the four SIDEDISTs can be replaced by a single field holding the radius of the block. This is because the SIDEDISTs of the BLACK nodes are all equal to the sum of the DIST value and the radius of the block. If these changes are made, assuming that all three node types are equally likely there is an average of two fields per node required to compute the distances, as opposed to five in the naive storage method used in the algorithm. To enhance the clarity of the algorithm, this was not done.

The major difference between the distance transform used by Samet and that used here is that Samet allowed diagonal steps in the path, whereas only horizontal and vertical steps are allowed in the transform discussed here. The current algorithm could easily be modified to calculate the alternative distance, at the cost of increasing the amount of storage at each node, and visiting more neighbors at each node. Suppose that four CORNERDISTs, analogous to SIDEDISTs, are maintained at each node. By interrogating these nodes as well as the SIDEDISTs, the shortest distance using eight neighbors is obtained (which is equivalent to the chessboard distance) instead of that using four neighbors. Note that this modification

simplifies the algorithm when a BLACK node is processed. This is because not all the eight neighbors of the BLACK node can be BLACK, or the node would have been part of a larger BLACK node. Storage in this case can be reduced to about three fields per node.

Notice that the shortest path to a WHITE border does not necessarily lead to the closest WHITE neighbor. For example, a WHITE block diagonally adjacent to a BLACK block (e.g. its NE neighbor) may be on a path longer than another WHITE block that does not touch the BLACK block at all (see Figure 4). This can occur if both the N and E blocks adjacent to the BLACK block are large. In Figure 4, the distance from A to B is 9 units, while that from A to C is only 2 units. In consequence, this transform is very sensitive to shifts in the quadtree origin. If shifts are to be considered, the transform of Samet, which depends only on the endpoints of the path, should be used.

Another difference between the algorithm presented here and that of Samet is in their treatement of border nodes. Samet treated a node on the border as being adjacent to a WHITE node; here the border is (effectively) considered to be BLACK. The distance of a border node is calculated along a path to an existing WHITE node. Should no such node exist (i.e. if the whole image is BLACK), the distance is undefined.

In the introduction it was mentioned that one of the reasons for calculating region-to-border distances is to enable thinning operations to be carried out. It should be pointed out, however, that this would not be a sensible application in a quadtree representation. Quadtrees are only cost-effective when the average node size is large compared to the number of nodes, whereas thinning is appropriate for elongated objects in which the number of nodes is large compared to their size. It would seem to be better to describe thinned objects in terms of generalized ribbons (Brooks *et al.* [1]). Other applications of the distance transform, such as the medial axis transform [6], may still be applicable to a quadtree representation.

7. Conclusions

An algorithm has been presented for computing a path-length distance transform for a binary image represented by a quadtree. The algorithm employs a top-down traversal of the tree, and calculates, in a single pass, the distance from the center of each BLACK node to the border of its nearest WHITE neighbor.

The algorithm was shown to have an average worst-case execution time proportional to the product of the log of the image diameter and the number of nodes in the tree.

The algorithm was compared with that of Samet [13] who used a different distance transform. It was seen to require slightly less time, but to use more space than that algorithm. It would be interesting to discover algorithms for computing distances using the more common Euclidean and city block metrices, but these are not as readily applied to the quadtree representation.

References

1. R. A. Brooks, R. Greiner, and T. O. Binford, A model based vision system, Proc. Image Understanding Workshop, May 1978, pp. 36-44.

2. C. R. Dyer, Computing the Euler number of an image from its quadtree, Computer Science TR-769, University of Maryland, College Park, Maryland, May 1979.

3. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, IEEE Trans. PAMI-7, 1979, pp. 145-153.

4. G. M. Hunter and K. Steiglitz, Linear transformations of pictures represented by quadtrees, Computer Graphics Image Processing 10, 1979, pp. 289-296.

5. A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics Image Processing 5, 1976, pp. 68-105.

6. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

7. H. Samet, Region representation: quadtrees from boundary codes, Computer Science TR-741, University of Maryland, College Park, Maryland, March 1979.

8. H. Samet, Computing perimeters of images represented by quadtrees, Computer Science TR-755, University of Maryland, College Park, Maryland, April 1979.

9. H. Samet, Connected component labeling using quadtrees, Computer Science TR-756, University of Maryland, College Park, Maryland, May 1979.

10. H. Samet, Region representation: raster-to-quadtree conversion, Computer Science TR-766, University of Maryland, College Park, Maryland, May 1979.

11. H. Samet, Region representation: quadtrees from binary arrays, Computer Science TR-767, University of Maryland, College Park, Maryland, May 1979.

12. H. Samet, Region representation: quadtree-to-raster conversion, Computer Science TR-768, University of Maryland, College Park, Maryland, June 1979.

13. H. Samet, A distance transform for images represented by quadtrees, Computer Science TR-780, University of Maryland, College Park, Maryland, July 1979.

14. M. Shneier, Linear time calculations of geometric properties using quadtrees, Computer Science TR-770, University of Maryland, College Park, Maryland, May 1979.

	node	caller	calls	DIST	SIDEDISTS
1	a	-	b,c,d,e	0	0,0,0,0
2	b	a	7,f,g,22	0	0,0,0,0
3	1	b	-	0	0,0,0,0
4	f	b	2,3,4,5	0	0,0,1,0
5	2	f	-	0	0,0,0,0
6	3	f	-	0	0,0,0,0
7	4	f	-	1/2	1,1,1,1
8	5	f	-	1/2	1,1,1,1
9	g	b	6,7,8,9	0	0,0,0,1
10	6	g	-	0	0,0,0,0
11	7	g	-	1/2	1,1,1,1
12	8	g	-	0	0,0,0,0
13	9	g	-	1/2	1,1,1,1
14	22	b	j,i	2	3,3,3,3
15	j	22	12,14,11,13	0	1,0,0,0
16	12	j	10,11	1/2	1,1,1,1
17	10	12	-	0	0,0,0,0
18	11	12	-	0	0,0,0,0
19	14	j	13	1/2	1,1,1,1
20	13	14	-	0	0,0,0,0
21	i	22	21,20,18,17	0	1,1,0,0
22	21	i	16,18,20	$1\frac{1}{2}$	2,2,2,2
23	16	21	14,15	1/2	1,1,1,1
24	14	16	-	0	0,0,0,0
25	15	16	-	0	0,0,0,0
26	18	21	17	1/2	1,1,1,1
27	17	18	-	0	0,0,0,0

Table 1

	node	caller	calls	DIST	SIDEDISTS
28	20	21	19	1/2	1,1,1,1
29	19	20	-	0	0,0,0,0
30	c	a	h,24	0	0,0,0,0
31	h	c	23	0	0,0,0,0
32	23	h	-	0	0,0,0,0
33	24	c	-	0	0,0,0,0
34	d	a	25,26	0	0,0,0,0
35	25	d	-	0	0,0,0,0
36	26	d	-	0	0,0,0,0
37	e	a	k,30,32,33	0	0,0,0,0
38	k	e	28,31,27,29	0	0,0,0,0
39	28	k	27	1/2	1,1,1,1
40	27	28	-	0	0,0,0,0
41	31	k	29,30	1/2	1,1,1,1
42	29	31	-	0	0,0,0,0
43	30	27	-	0	0,0,0,0
44	32	e	-	0	0,0,0,0
45	33	e	-	0	0,0,0,0

Table 1 (continued)

The quadrants are numbered in the order in which their distances are established

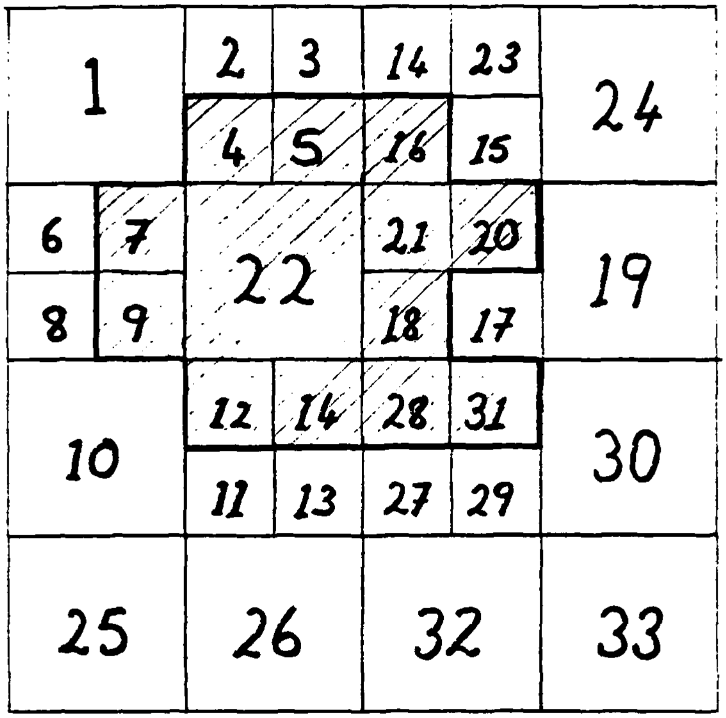

Figure 1a

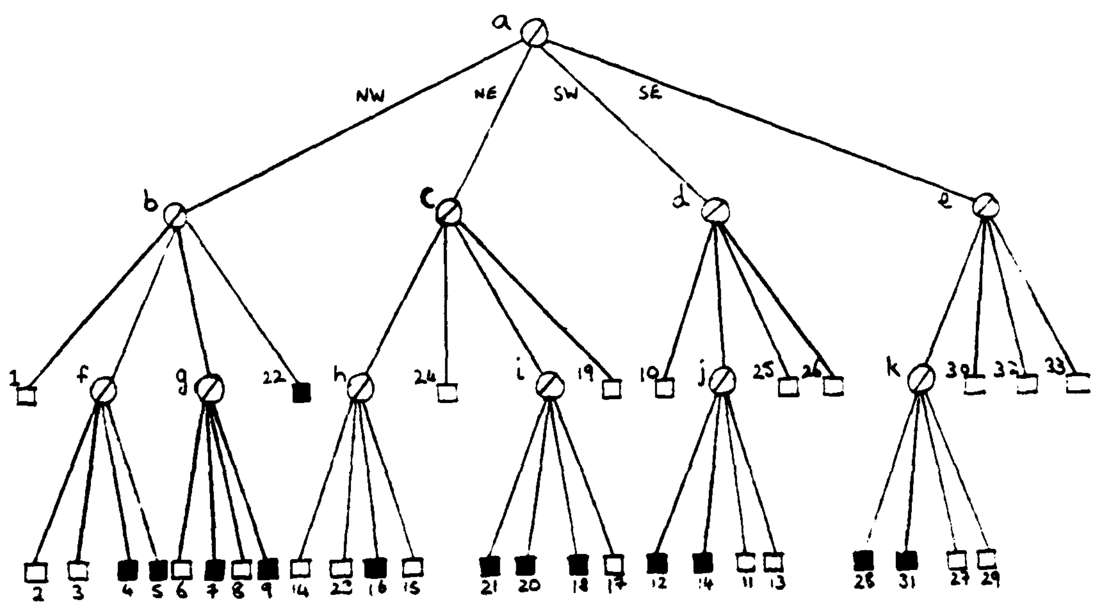

Figure 1b

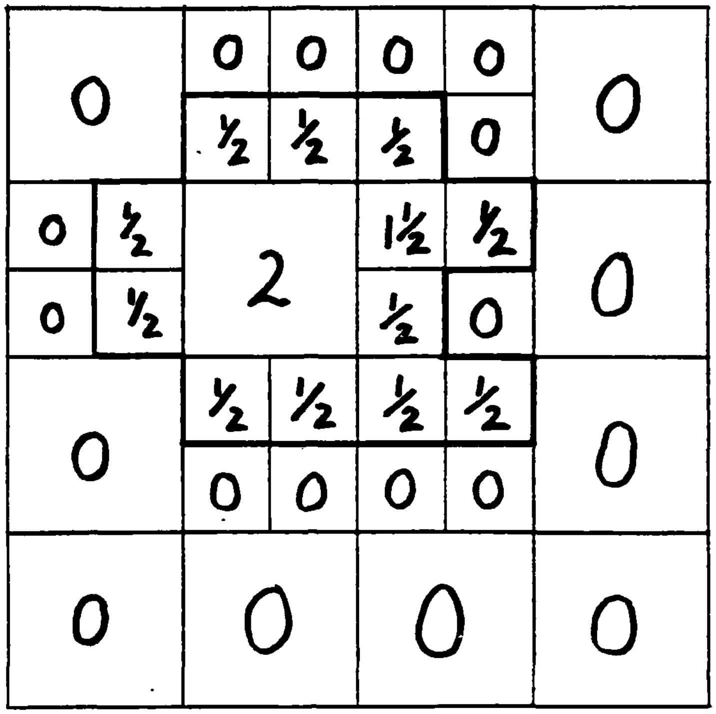

Figure 1c. Distances of the leaf nodes of Figure 1a.

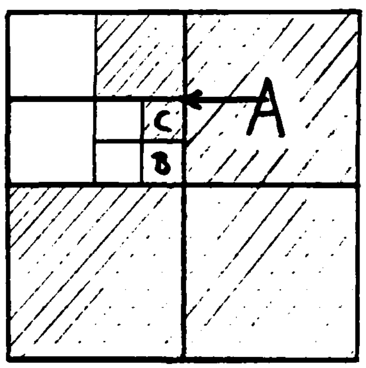

Figure 2

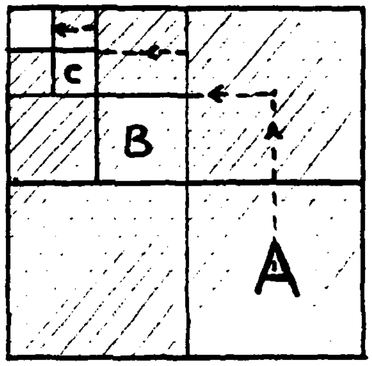

Figure 3

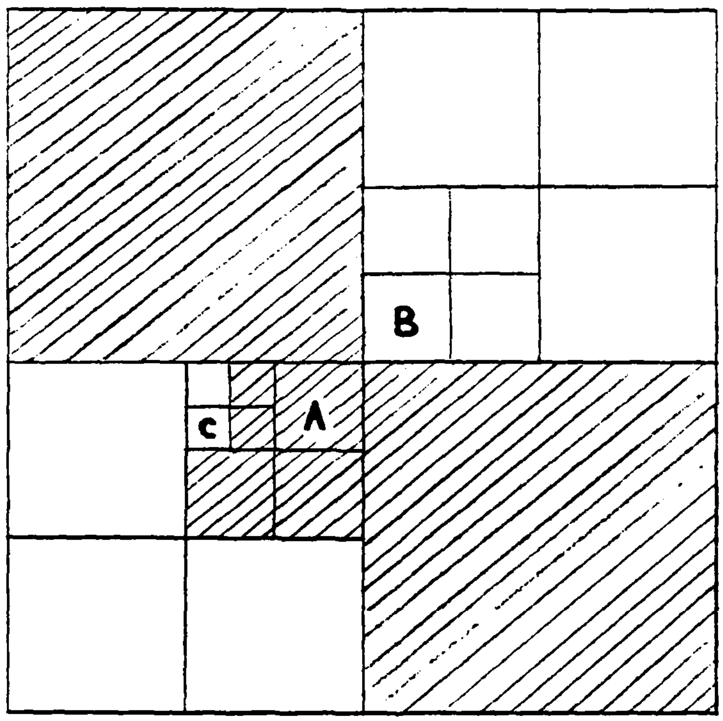

Figure 4

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A084 293	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A PATH-LENGTH DISTANCE TRANSFORM FOR QUADTREES.		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-794
7. AUTHOR(s) Michael Shneier		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138, DARPA Order-3206
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Night Vision Laboratory Ft. Belvoir, VA 22060		12. REPORT DATE July 1979
		13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Distance transforms
Quadtrees

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A distance measure is defined for a quadtree representation of a binary image. An algorithm is presented that calculates the distance from the center of each BLACK node to the border of the nearest WHITE node. The distance is defined as the path length from the center of the BLACK node, through the center of intervening BLACK nodes, to the WHITE border. The worst case average execution time is shown to be proportional to the product of the log of the image diameter and the number of blocks in the image.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411 074